Burning Problem using ML

Imitation Learning for GNNs

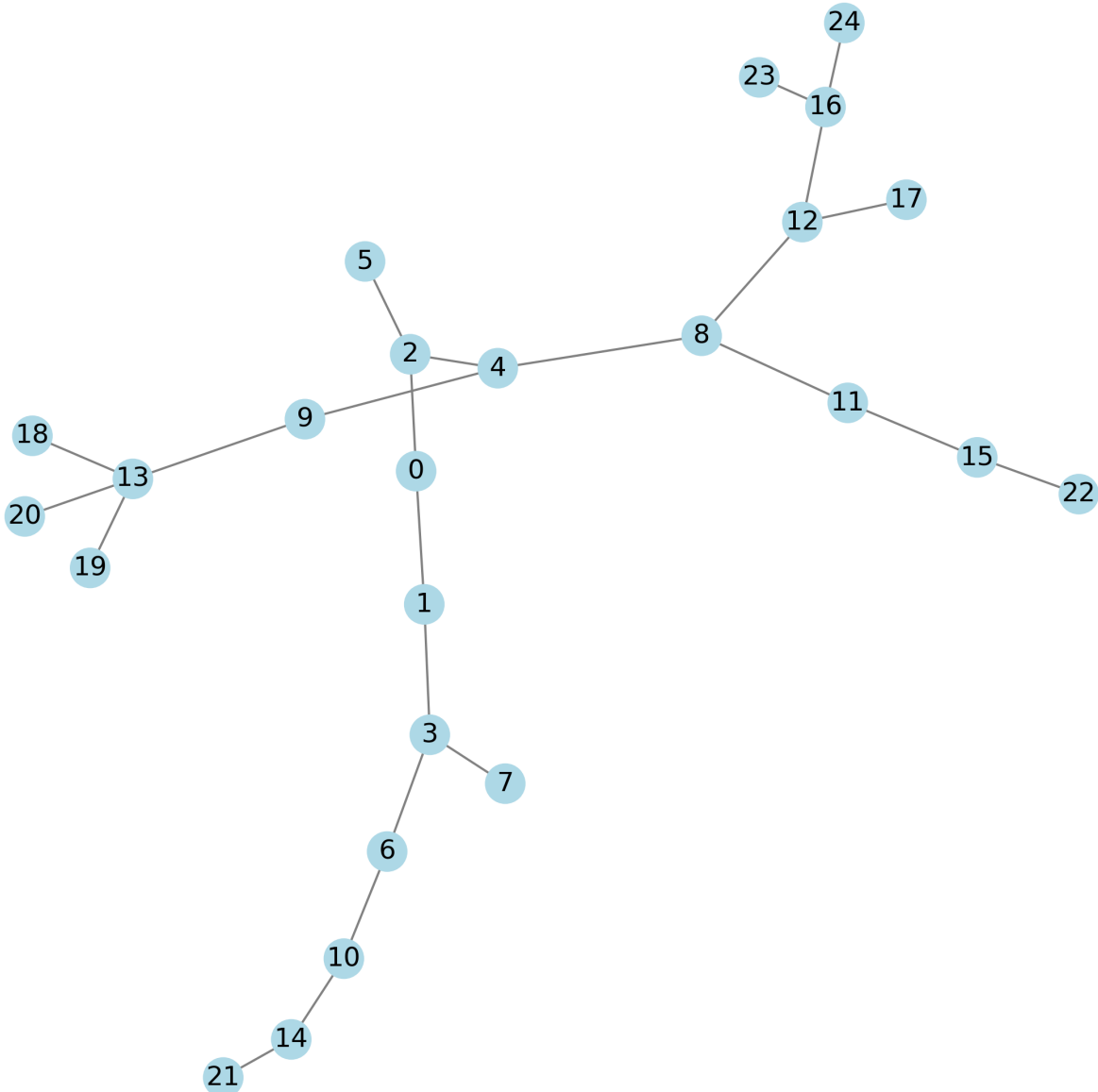
December 4, 2025

Introduction

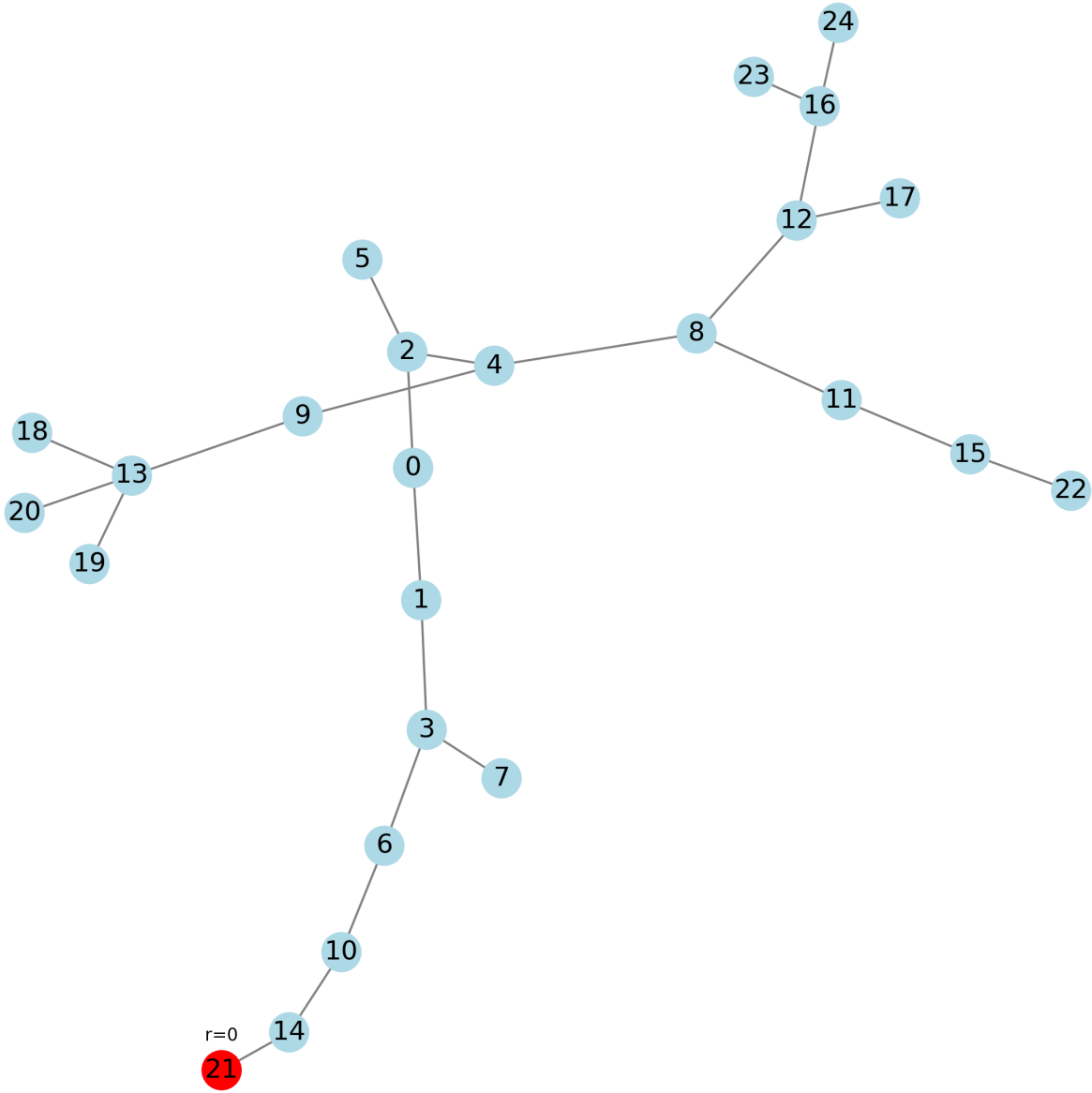
The Burning Problem

- **Graph Burning Problem**: equivalent to the **Graph Covering Problem**:
  - A ball of radius r around a node covers all nodes of distance $\leq r$.
  - The goal is to determine the shortest sequence of balls to cover the entire graph.
- **Burning number** = minimum number of balls used to burn the graph.

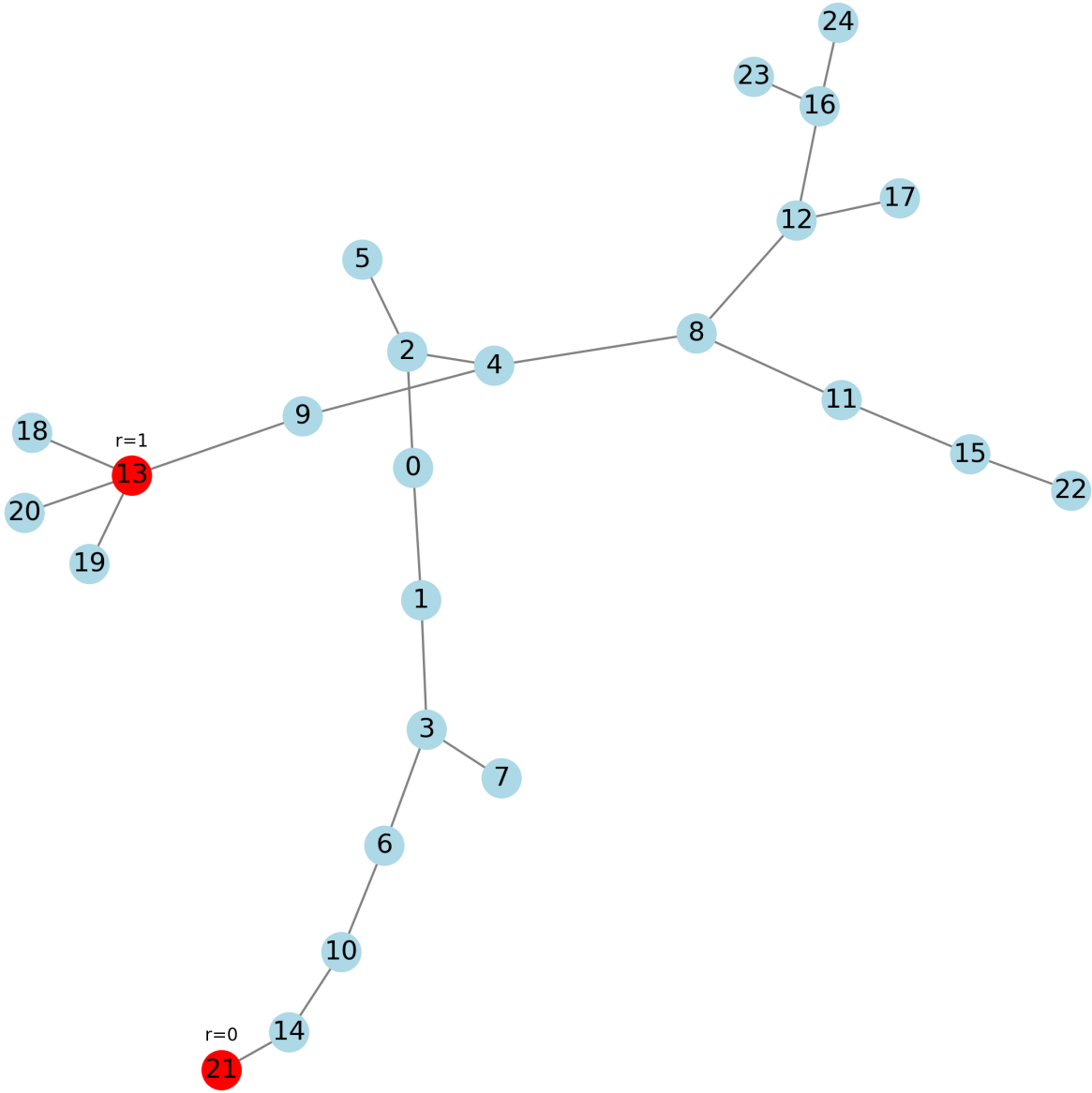
Step 0:



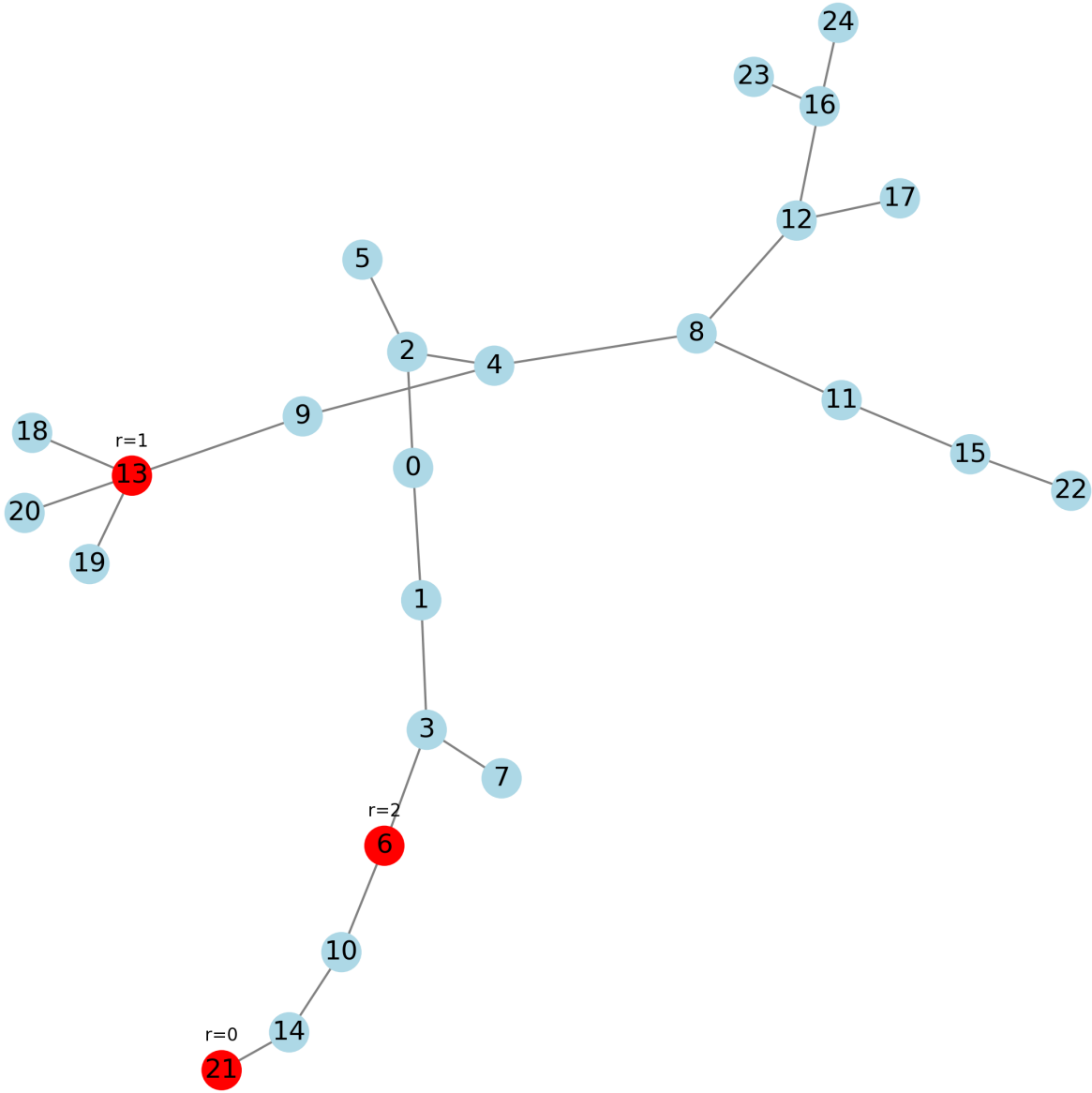
Step 1:



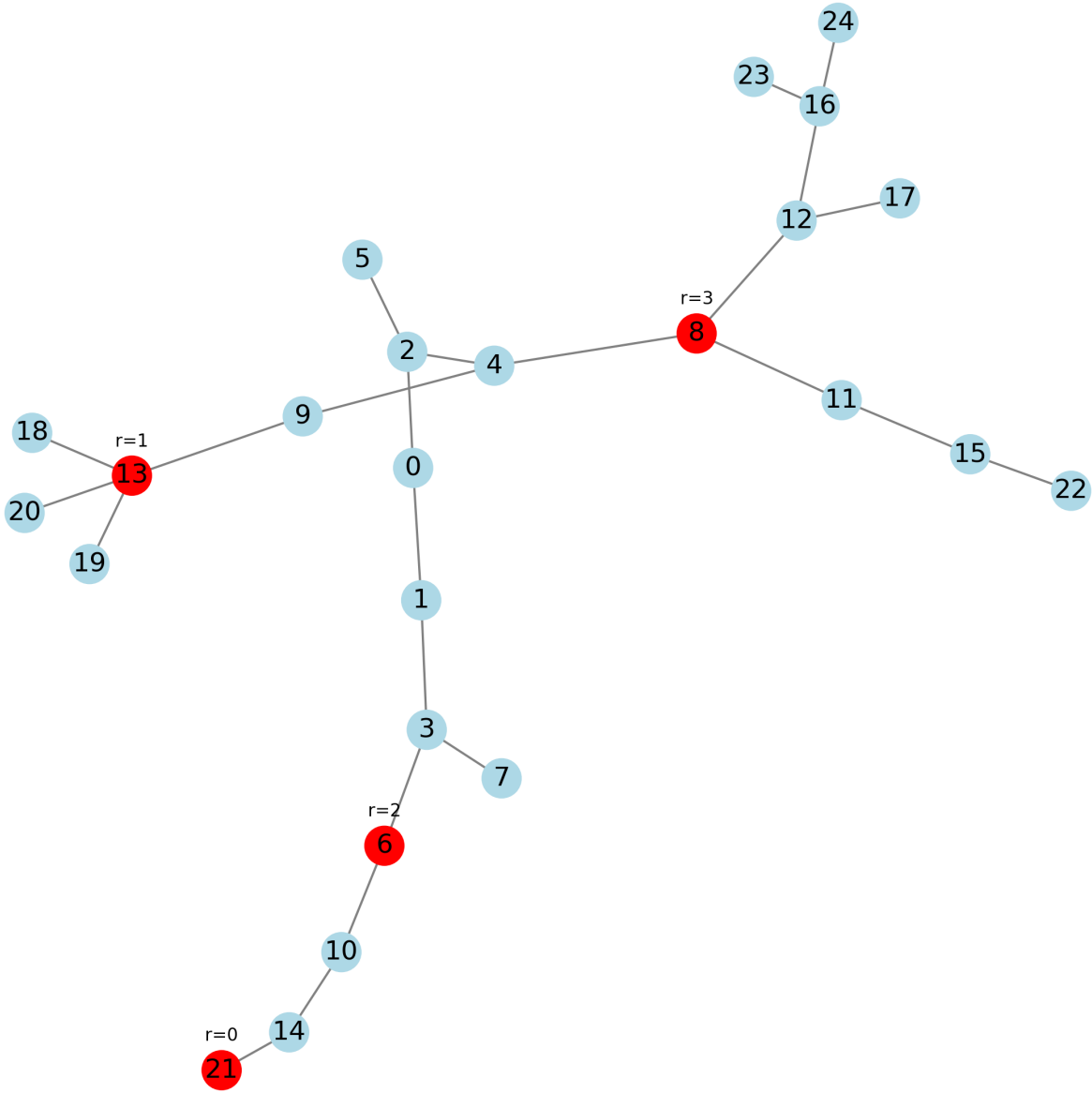
Step 2:



Step 3:



Step 4:



Motivation

- **NP-hard problem**: Finding the burning number is computationally intractable for large graphs.
- Optimal solutions are hard to find, so we often rely on heuristics – it would be nice to have a way to reliably find good coverings.
- **Real-world relevance**: models spread of information, diseases, or influence in networks.

Problem Formulation

- **Problem**: Can we train a model to efficiently find a minimal solution to the **Burning Problem**?
- **Inspiration**: Prior work showed success on another NP-hard task (finding Hamiltonian cycles using graph neural networks).
- **Objective**: We aim to apply a similar approach (imitation learning + GNN) to the graph burning problem.

Problem Simplification

- We simplified the problem to covering trees.
- Trees are a practical choice as they are easy to generate uniformly and they are the hardest to cover in practice (not a well connected network).
- Furthermore, they have good generalization property since to cover a graph we are only required to cover one of its spanning trees.

Data Generation

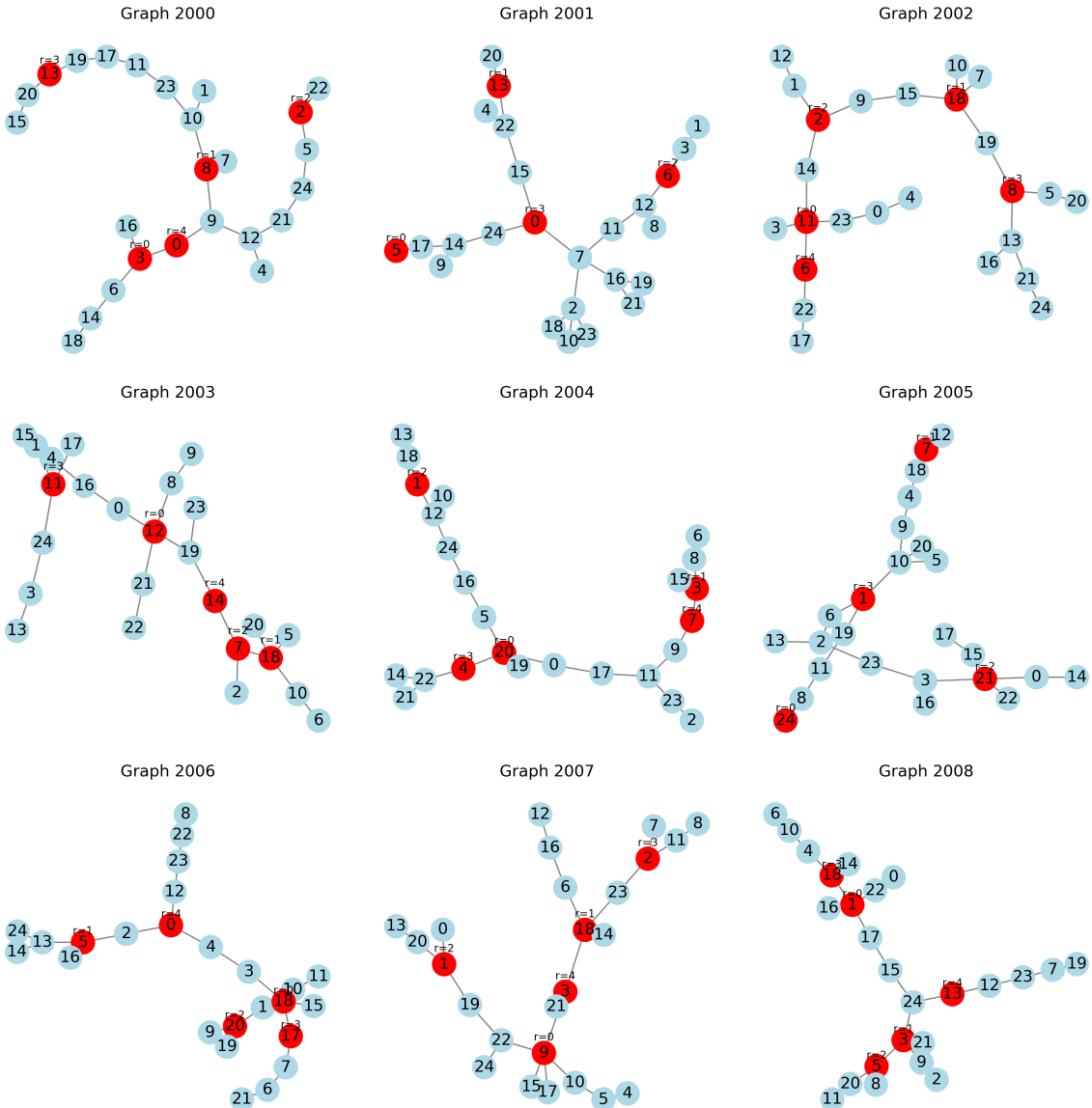
Data Generation

- We decided to generate our own dataset to have more control over its properties and ensure that the training set is well representative of general network structures.
- **Uniform random trees** generated via *Prüfer sequences*: each random Prüfer code of length $n-2$ yields a uniformly random labeled tree. These sequences can be easiliy generated and decoded back to there tree representation.

Methodology

- For each tree, computed the exact **burning number** by brute force:
  - Tried all possible graph covers and chose a minimal one arbitrarily.
  - Exponential search – very slow: computing optimal burns for 5,000 trees took over 10 hours.
- These optimal sequences (node orderings) serve as expert demonstrations for training.

Examples


Graph 2000
Graph 2001
Graph 2002
Graph 2003
Graph 2004
Graph 2005
Graph 2006
Graph 2007
Graph 2008

Our Model

Imitation Learning Approach

- We adopt **imitation learning**: train a policy to mimic an expert (the minimal covering).
- Formulate as a sequential decision process:
  - State = graph + inherit graph property h + partial solution $\mathbf{x}$
  - Action = estimate the probabilities p of each node being the next center and pick the node c with largest associated probability.
  - Expert policy: compare with the cover label and punish the model by $-\ln(p[c])$. We reinitialize $\mathbf{x}$ with the correct partial solution and update h with model output.

GNN and message passing layers

- Nodes exchange information with neighbors through message passing layers, aggregating structural context.
- After L layers, each node's embedding captures information from its L-neighborhood (graph structure + covers).
- This is the entire essence of the model and how we may hope to correctly choose optimal centers.

Algorithm

Input:

$$\begin{cases} G & \text{graph with } n \text{ vertices} \\ x \in \mathbb{R}^{n \times d_{\text{in}}} & \text{partial solution representation} \\ h \in \mathbb{R}^{n \times d_h} & \text{latent features} \end{cases}$$

Output:

$$p \in [0,1]^n \quad \text{next-step probabilities per node}$$

Hyperparameters:

$$d_{\text{in}} = 2, \quad d_h = 32, \quad n_p = 4$$

Parameters:

$$\theta \equiv \{W_E, b_E, W_P, b_P, \ldots\} \quad \text{neural network weights}$$

Forwarding

Algorithm:

Encoder - Initialize features:

$$z_i = W_E(x_i \oplus h_i) + b_E \in \mathbb{R}^{d_h}, \quad h_i = z_i$$

Message-passing - Apply residual max-MPNN layers:

$$\text{for } k = 1 \text{ to } n_p : \quad \begin{cases} m_i = \max_{j \sim i} \mathrm{ReLU}\left(W_M^{(k)}(h_i \oplus h_j) + b_M\right) \in \mathbb{R}^{d_h} \\ h_i = h_i + \mathrm{ReLU}\left(W^{(k)}(h_i \oplus m_i) + b^{(k)}\right) \in \mathbb{R}^{d_h} \end{cases}$$

Decoder - Extract logits and probabilities:

$$l_i = W_D(z_i \oplus h_i) + b_D \in \mathbb{R}, \quad i = 1, \ldots, n$$

$$\text{if i is already a center}, \quad l_i = -\infty$$

$$p = \mathrm{softmax}(l) \in \mathbb{R}^n$$

GNN Architecture

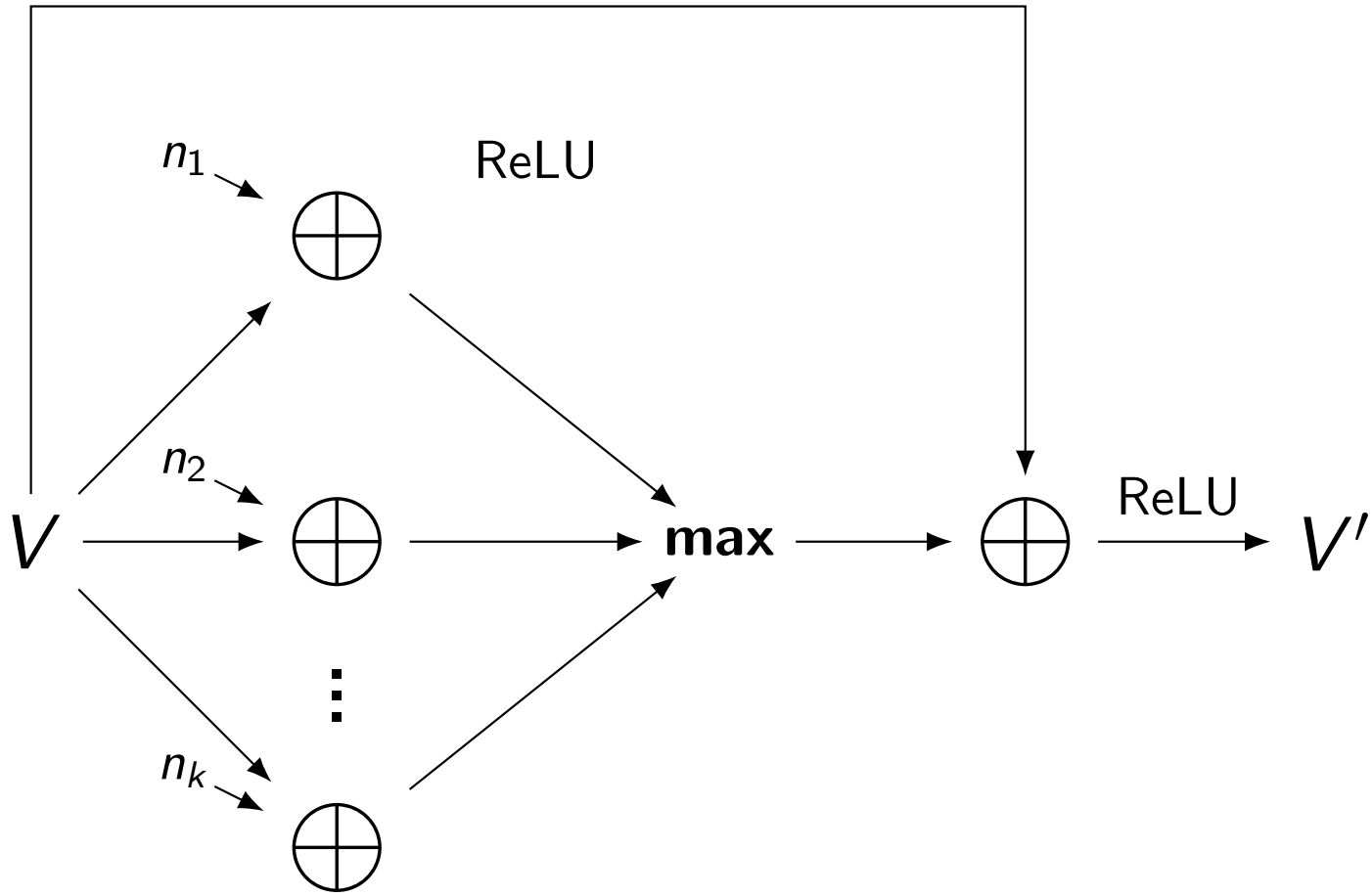


Figure: MP layer 1-node

Training

Training and Experiments

- Training via supervised learning (behavior cloning): minimize error between predicted next-node and expert's choice.
- We used Adam optimizer with steps of size 10^{-3} for 100 epochs.

Training Algorithm

```
1: for epoch = 1 to max do
2:     for tree,cover ∈ D do
3:         h = Initializeh(tree)
4:         loss = 0
5:         for k = 1 to len(cover) do
6:             x = EncodeCoverk(cover,k)
7:             c = center of k-th ball
8:             p,h = model(tree,x,h)
9:             loss += -ln(p[c])
10:        end for
11:        backprop(loss)
12:    end for
13: end for
```

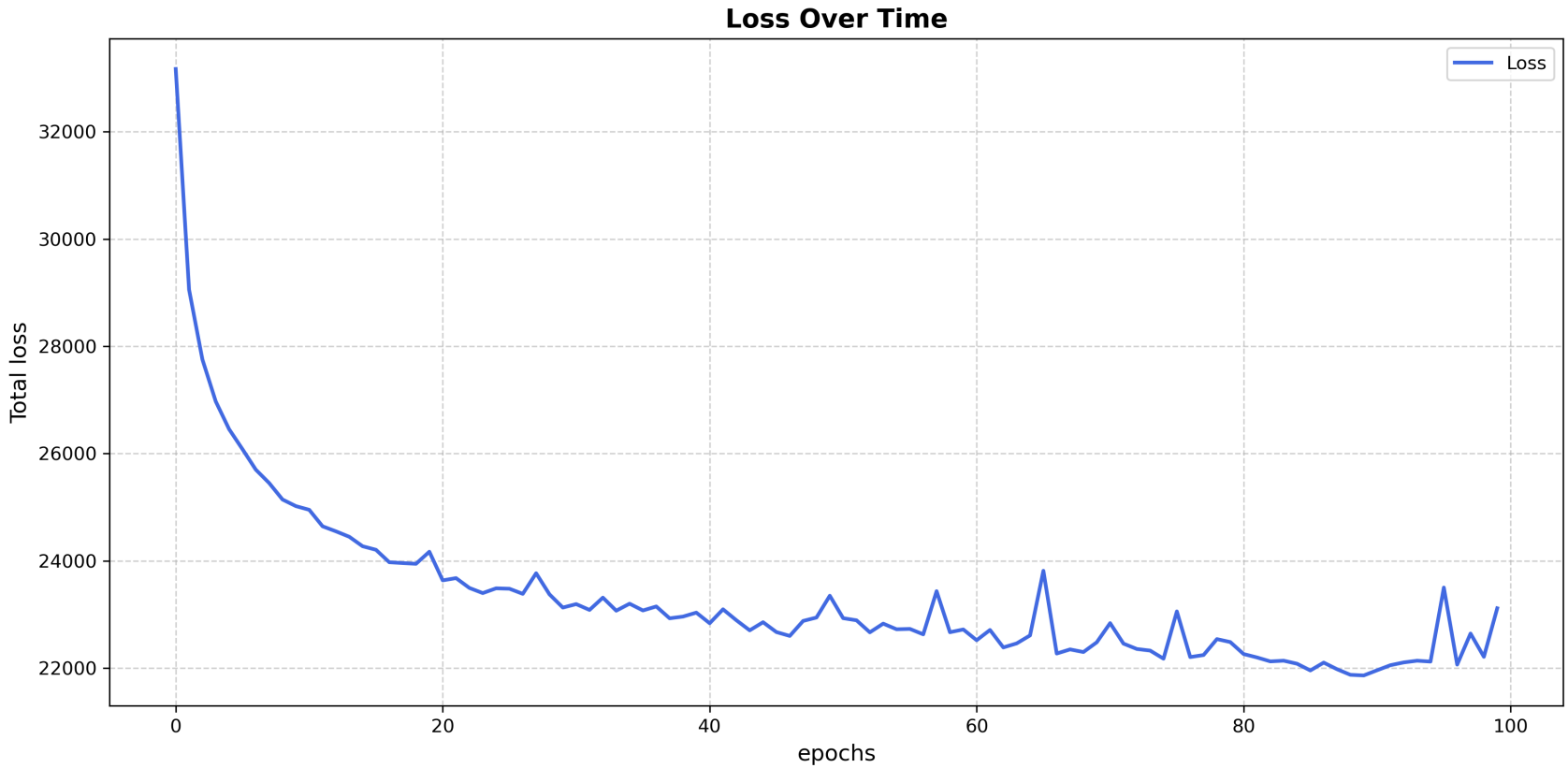
Improving Training

- An important aspect of the model is the initialization of h. We tested multiple different approaches which include adding different structural graph properties, initializing to a zero matrix and adding random noise to the matrix.
- The optimal and most intuitive result was to initialize h to be a column of the node degrees and eigenvectors of the Laplacian while completing to $n \times 32$ matrix with zeros.

Improving Training

- We also experimented on the number of message passing layers, we concluded that 4 was optimal. This is also intuitively correct since the balls we use in the coverings are (conjectured to be) at most of size 4.
- Pool function choice, sum, average and max.

Loss Over Time



Results

Evaluation Procedure

- After training, we evaluate the model on 1,000 unseen trees. For each test tree, we retrieve the optimal burning number k computed by brute-force.
- We run the model iteratively k times, updating the input at each step by adding the ball chosen by the model previously.
- After k iterations, we verify if the balls placed cover the entire graph, indicating a valid solution.

Performance and Success Criteria

- The model is **not** evaluated by reproducing the exact brute-force solution.
- Success means the solution covers the graph using exactly k balls.
- Our model achieves a success rate of **22%**.
- Randomly sampling k min centers yields only about **0.3%** success.

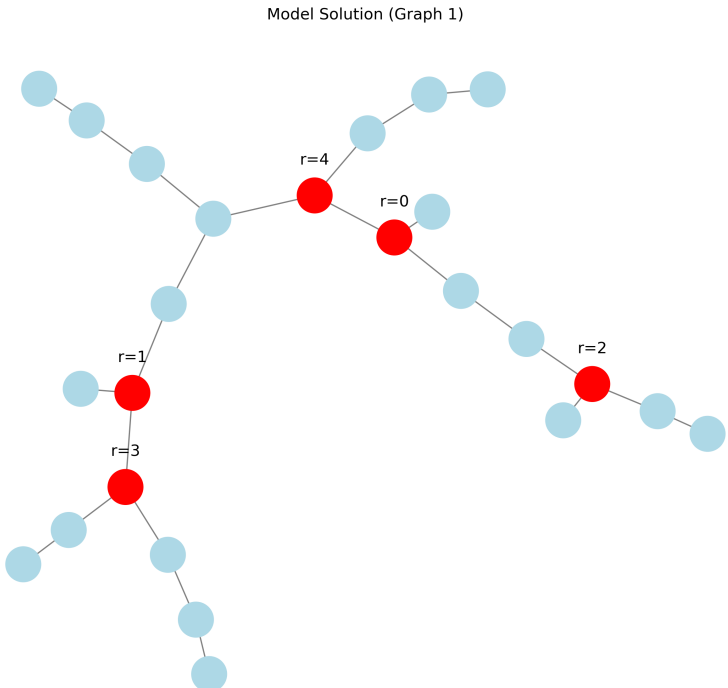
Method	Success Rate
Random Sampling	0.3%
Our Model	22%

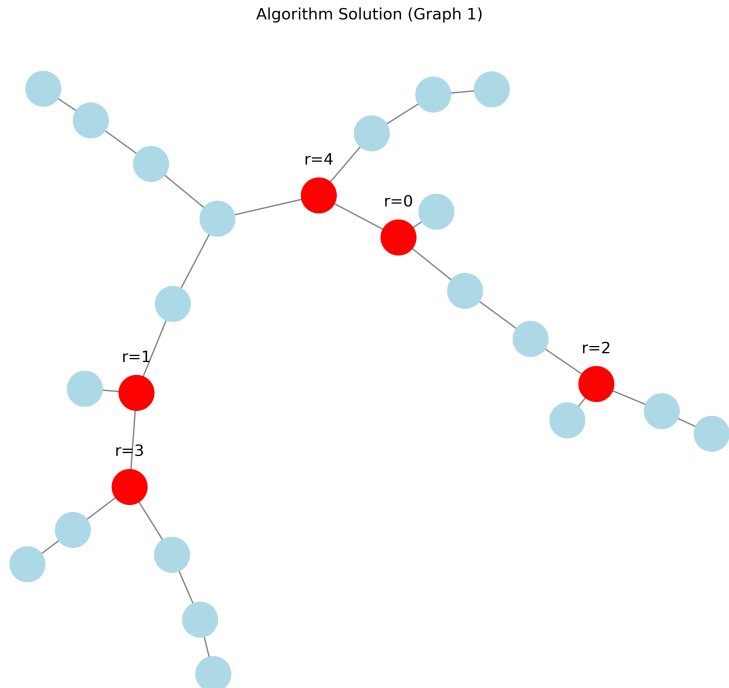
Table: Success rate comparison

Interpretation and Generalization

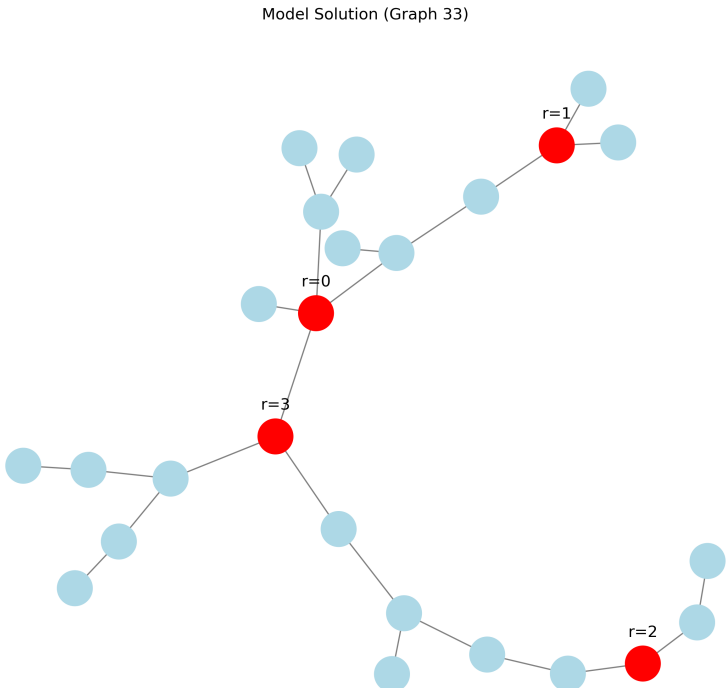
- Solutions from our model and brute-force are rarely identical but show strong similarities.
- This indicates the model has **actually** learned the underlying problem structure.
- Rather than *simply* imitating expert solutions, the model generalizes well.

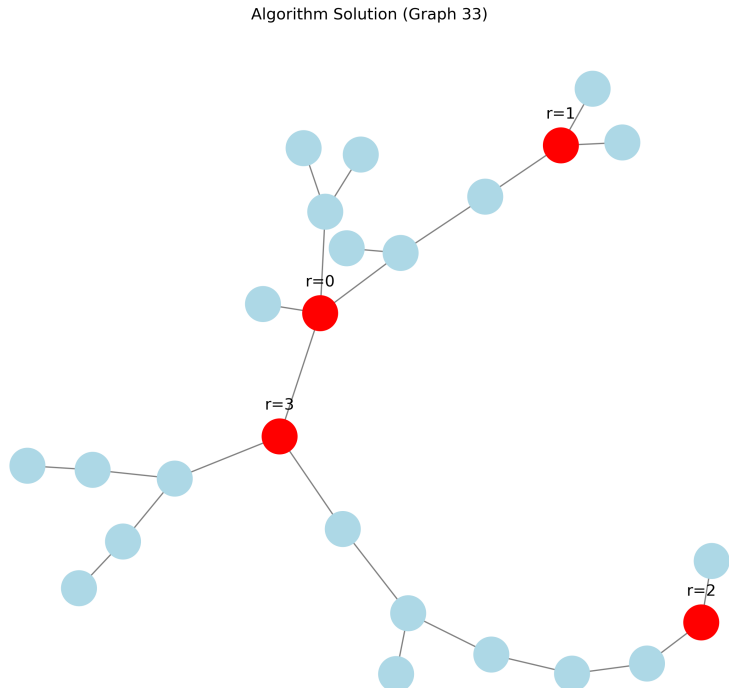
Brute Force vs Model solutions (1)



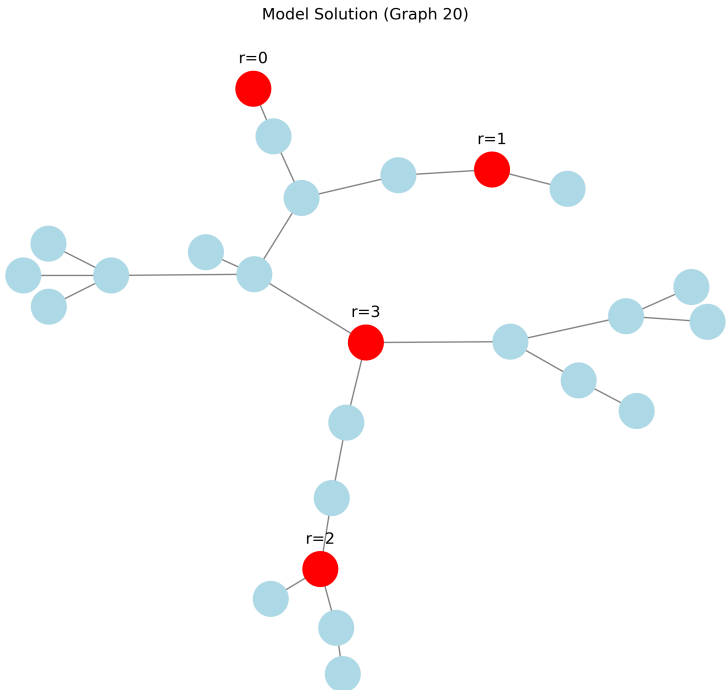


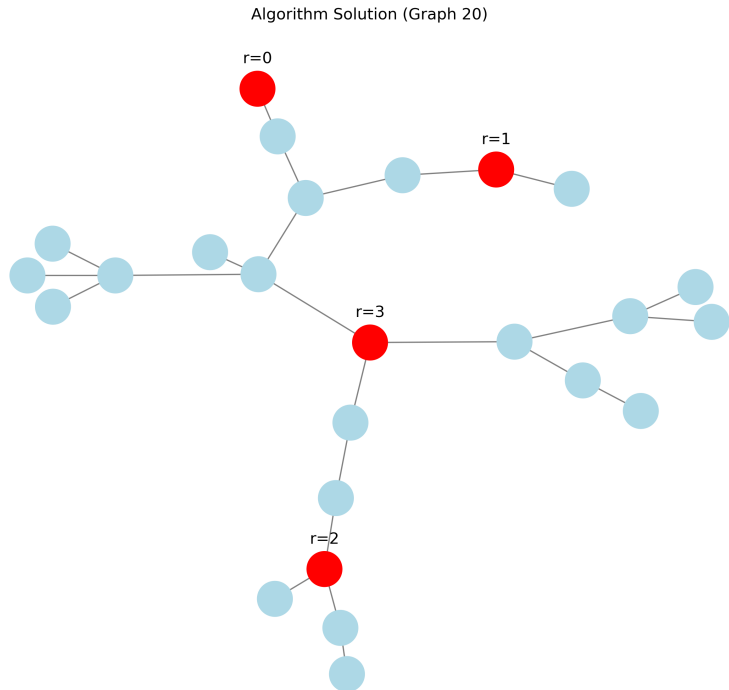
Brute Force vs Model solutions (2)



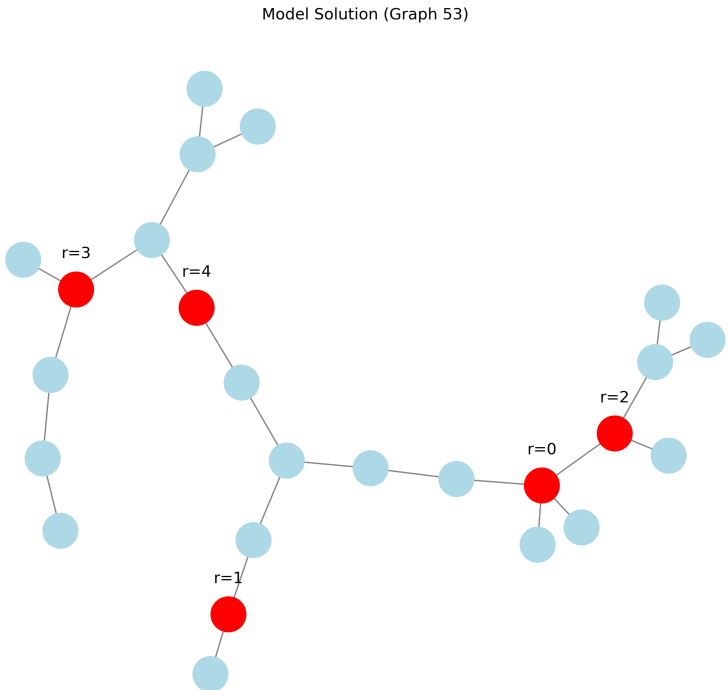


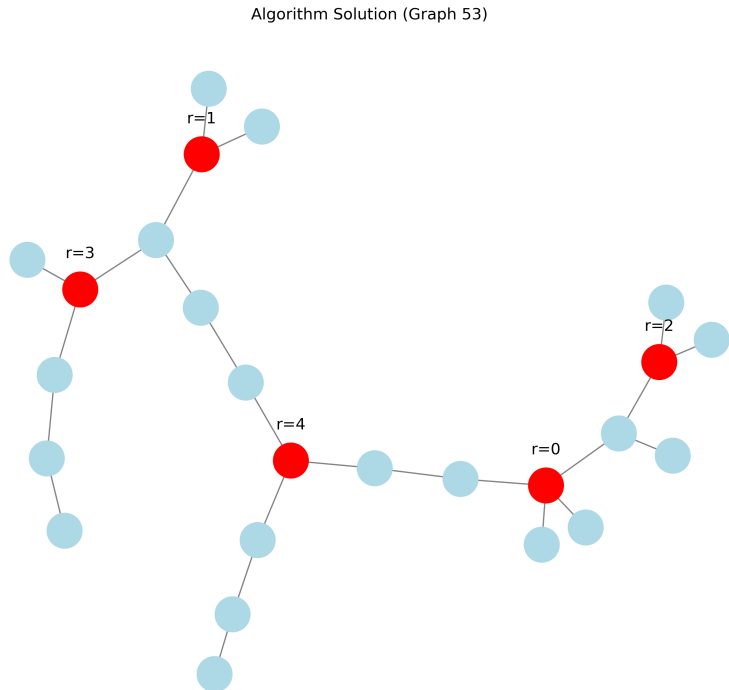
Brute Force vs Model solutions (3)



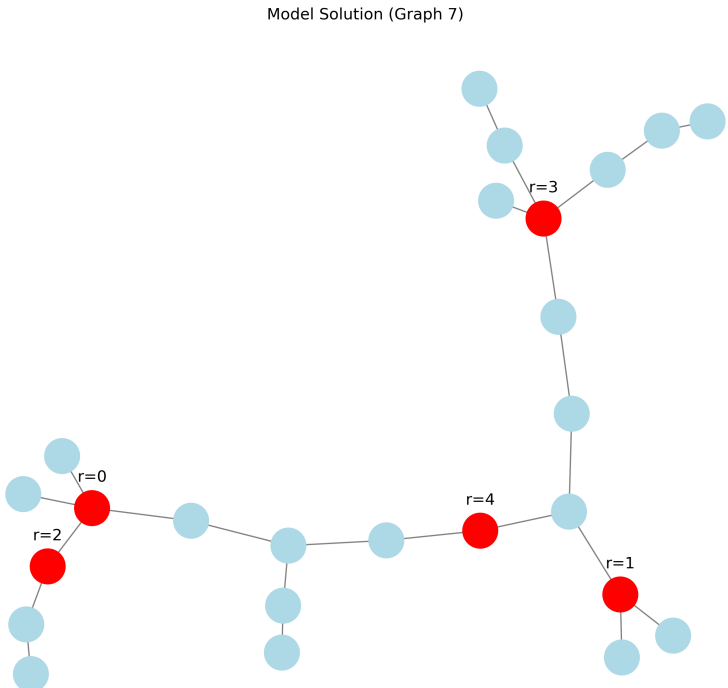


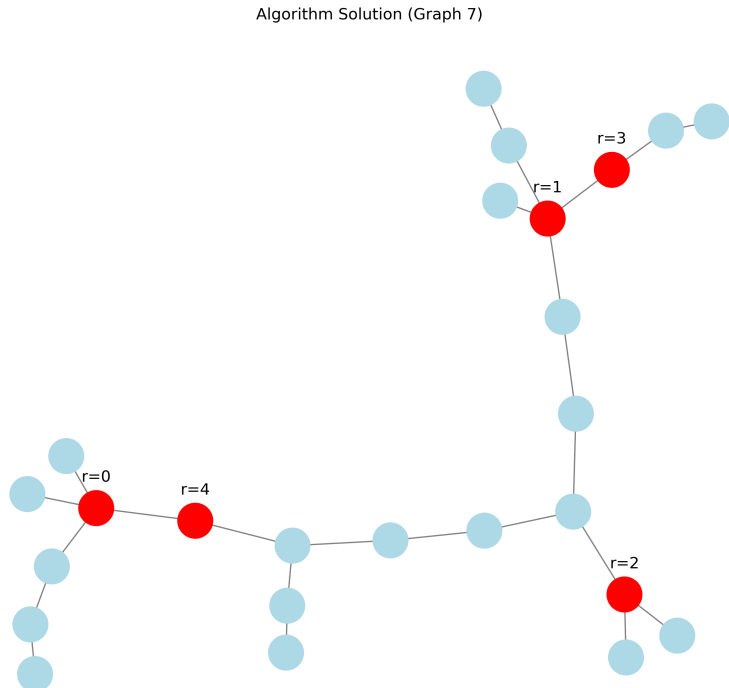
Brute Force vs Model solutions (4)





Brute Force vs Model solutions (5)





Conclusion

Conclusion & Key Results

Goal: Solve the NP-hard **Graph Burning Problem** using GNNs and imitation learning.

- **Approach**:
  - Trained a GNN to mimic brute-force solutions on **uniform random trees**.
  - Used message-passing to iteratively select burning centers.
- **Results**:
  - Model achieved **22% success rate** (vs. **0.3%** for random sampling).
  - Learned to generalize: solutions were structurally similar to brute-force (though not identical).

Discussion 1: Performance Gap with Reference Model

- Reference model finds Hamiltonian cycles in **75% of cases** for graphs of size 25, vs **22%** for us.
- Our model underperforms due to key differences:
  - **Nature of the problem:**
    - Hamiltonian cycle task has consistent output at each step.
    - Our task: ball radius increases with each step $\Rightarrow$ changing output.
  - **Training becomes harder** due to the shifting objective.
  - **Fewer model calls per graph:**
    - Hamiltonian: 25 calls/graph (one per node).
    - Our task: typically ≤ 4 calls/graph.
  - **Less opportunity to learn latent structure.**

Discussion 2: Next Steps and Generalization

- Inspired by the Hamiltonian model's scalability to larger graphs, we could evaluate our model's generalization ability.
- Proposed experiment:
  - Generate a dataset of 1,000 trees, each with 50 nodes.
  - Use brute-force to compute **ground truth** solutions.
  - Measure model's success rate in matching optimal outputs.
- **High success rate** would:
  - Indicate strong learning and generalization.
  - Validate our model beyond its training data.

$\rightarrow$ Was not possible for us given the enormous time necessary to compute brute-force solutions.

References

Bonato, A., Janssen, J., & Roshanbin, E. (2014). *Burning a Graph as a Model of Social Contagion*. Algorithms and Models for the Web Graph (WAW 2014), LNCS 8882, pp. 13–22.

Bosnić, F., & Šikić, M. (2023). *Finding Hamiltonian cycles with graph neural networks*. arXiv:2306.06523.